HY-225: Οργάνωση Υπολογιστών
Άνοιξη 2018

Τμ. Επ. Υπολογιστών

Άσκηση 7:
Πρόγραμμα Συνδεδεμένης Λίστας και Διαδικασιών

Προθεσμία έως Πέμπτη 15 Μαρτίου 2018, ώρα 23:59 (βδ. 6.2) (από βδ. 4.3)

Υπενθύμιση Διαγωνισμού Προόδου: Σάββατο 17 Μαρτίου 2018, ώρα 12-2
(Ο βαθμός της εξέτασης Προόδου –όποιος και αν είναι αυτός– αποτελεί το 15 % του βαθμού μαθήματος, **οιασδήποτε** περιόδου)

7.1 Δομές Δεδομένων (Data Structures):

Σε αυτή την άσκηση θα χρησιμοποιήσουμε μία δομή δεδομένων (structure) που θα αποτελεί ένα κόμβο μιάς συνδεδεμένης λίστας (linked list). Κάθε κόμβος (δομή δεδομένων) μας θα αποτελείται από δύο λέξεις (των 32 bits καθεμία): έναν ακέραιο "data" που θα περιέχει την "πληροφορία χρήστη", κι έναν δείκτη σύνδεσης (pointer) "nxtPtr" που θα περιέχει τη διεύθυνση του επόμενου κόμβου στη λίστα (στον τελευταίο κόμβο της λίστας, nxtPtr=0). Τα δύο στοιχεία (λέξεις) της δομής μας θα βρίσκονται σε διαδοχικές θέσεις (λέξεις) της μνήμης. Επομένως, κάθε δομή (κόμβος) μας θα έχει μέγεθος 2x4=8 bytes. Διεύθυνση μιάς δομής είναι η διεύθυνση του πρώτου στοιχείου της, δηλαδή του στοιχείου με "μηδενικό offset", που γιά μας είναι το "data". Άρα, το δεύτερο στοιχείο της δομής μας, ο "nxtPtr", βρίσκεται στη διεύθυνση που προκύπτει προσθέτοντας 4x1=4 στη διεύθυνση της δομής (κόμβου).

7.2 Δυναμική Εκχώρηση Μνήμης (Dynamic Memory Allocation):

Το πρόγραμμά σας θα ζητάει και θα παίρνει δομές (κόμβους) από το λειτουργικό σύστημα "δυναμικά", την ώρα που τρέχει (σε run-time). Γιά το σκοπό αυτό θα χρησιμοποιήστε το κάλεσμα συστήματος (system call) "sbrk" (set break). Το κάλεσμα αυτό "σπρώχνει" πιό πέρα (προς αύξουσες διευθύνσεις μνήμης) το σημείο "break", το όριο δηλαδή πριν από το οποίο οι διευθύνσεις μνήμης που γεννά το πρόγραμμα είναι νόμιμες, ενώ μετά από το οποίο (και μέχρι την αρχή της στοίβας) οι διευθύνσεις είναι παράνομες και ενδεχόμενη χρήση τους προκαλεί το γνωστό από την C "segmentation violation - core dumped". Το κάλεσμα συστήματος "sbrk" έχει κωδικό 9, και περιγράφεται στις σελίδες A.44 - A.45 του παραρτήματος Α του βιβλίου, από παλαιότερη Αμερικανική έκδοση, που υπάρχει στο αρχείο **`~hy225/spim/documentation/HP_AppA.pdf`** και λειτουργεί κατ' αναλογία με τα άλλα καλέσματα συστήματος (εκτύπωσης και ανάγνωσης) που χρησιμοποιήσατε σε προηγούμενες ασκήσεις. Πριν το καλέσετε, θέτετε τον καταχωρητή $a0 ($4) να περιέχει το πλήθος των νέων bytes που επιθυμείτε (ένας αριθμός). Μετά την επιστροφή του, ο καταχωρητής $v0 ($2) περιέχει τη διεύθυνση του νέου block μνήμης, του ζητηθέντος μεγέθους, που το σύστημα δίνει στο πρόγραμμά σας (έναν pointer). Η επιστρεφόμενη διεύθυνση μνήμης είναι πάντα διάφορη του μηδενός (εκτός --πιθανότατα-- όταν γεμίσει όλη η μνήμη, αλλά δεν χρειάζεται εσείς εδώ να ελέγχετε κάτι τέτοιο), και είναι πάντα ευθυγραμμισμένη σε όρια λέξεων (πολλαπλάσιο του 4) (τουλάχιστο στη δική μας περίπτωση, που ζητάμε πάντα blocks μεγέθους πολλαπλάσιου του 4, αλλά --πιστεύω-- και σε κάθε περίπτωση).

Άσκηση 7.3: Κατασκευή και Σάρωση Συνδεδεμένης Λίστας

Γράψτε και τρέξτε στον SPIM, σε Assembly του MIPS, ένα πρόγραμμα που πρώτα θα κατασκευάζει και θα γεμίζει με θετικούς ακέραιους αριθμούς μία συνδεδεμένη λίστα (linked list), και στη συνέχεια θα την σαρώνει επαναληπτικά, τυπώνοντας κάθε φορά ένα διαφορετικό υποσύνολο των στοιχείων της --συγκεκριμένα: όσα στοιχεία της είναι μικρότερα από δοθείσα τιμή. Το πρόγραμμά σας θα κρατάει στον καταχωρητή $s0 (δηλαδή τον $16) τον pointer στην αρχή (στον πρώτο κόμβο) της λίστας, και θα αποτελείται από δύο κομμάτια, 7.4 και 7.5 (βλ. αμέσως παρακάτω). Στη συνέχεια, βασικά κομμάτια του προγράμματός σας θα τα κάνετε διαδικασίες (procedures), όπως περιγράφει η §7.6 (μερικές από τις διαδικασίες θα είναι υπερβολικά μικρές, αλλά αυτό γίνεται γιά λόγους εξάσκησης, ούτως ώστε το βάθος καλεσμάτων να φτάνει 2 επίπεδα κάτω από την main).

7.4: Κατασκευή της Λίστας. Χρησιμοποιήστε τον καταχωρητή $s1 ($17) σαν pointer στην ουρά (στον τελευταίο κόμβο) της λίστας. Γιά διευκόλυνση του βρόχου κατασκευής της λίστας (επειδή η εισαγωγή σε κενή λίστα διαφέρει από την εισαγωγή σε μη κενή λίστα), αρχικοποιήστε τη λίστα να περιέχει ένα αδιάφορο ("dummy") κόμβο: ένα κόμβο με data=0. Η αρχικοποίηση γίνεται ζητώντας και παίρνοντας ένα κόμβο από το λειτουργικό σύστημα, γράφοντας data=0 και nxtPtr=0 (τελευταίος κόμβος) σε αυτόν, και κάνοντας τους $s0 και $s1 να δείχνουν σε αυτόν τον κόμβο (να περιέχουν τη διεύθυνσή του). Μετά, μπείτε στο βρόχο ανάγνωσης στοιχείων και κατασκευής της λίστας. Σε κάθε ανακύκλωση αυτού του βρόχου:

1. Διαβάζουμε έναν ακέραιο αριθμό από την κονσόλα.
2. Εάν ο αριθμός αυτός είναι αρνητικός ή μηδέν, βγαίνουμε από το βρόχο.
3. Ζητάμε έναν νέο κόμβο από το λειτουργικό σύστημα (memory allocation).
4. Τοποθετούμε τον αριθμό που διαβάσαμε στο πεδίο "data" του κόμβου.
5. Συνδέουμε το νέο κόμβο στην ουρά της λίστας.

7.5: Σάρωση της Λίστας. Το δεύτερο μέρος του προγράμματος θα διαβάζει έναν μη αρνητικό αριθμό, και θα τυπώνει, με τη σειρά από την αρχή μέχρι το τέλος, όσα στοιχεία της λίστας είναι **μικρότερα** από αυτόν τον αριθμό. Μην χρησιμοποιήστε τον pointer στον τελευταίο κόμβο της λίστας (από την παλιά τιμή του καταχωρητή $s1 ($17)) γιά να βρίσκετε πού τελειώνει η λίστα --χρησιμοποιήστε τον nxtPtr κάθε κόμβου γιά να ξέρετε αν υπάρχει ή όχι επόμενος κόμβος στη λίστα. Το μέρος αυτό του προγράμματος κάνει τα εξής:

1. Διαβάζει έναν ακέραιο αριθμό από την κονσόλα και τον φυλάει στον καταχωρητή $s1 ($17). Εάν ο αριθμός αυτός είναι αρνητικός, το πρόγραμμα τερματίζει (βγαίνουμε από την main()).
2. Αρχικοποιεί τον καταχωρητή $s2 ($18) σαν δείκτη (pointer) σάρωσης, να δείχνει στον πρώτο κόμβο της λίστας (τον ξέρουμε από τον $s0 ($16)).
3. Μπαίνει σ' ένα βρόχο, σε κάθε ανακύκλωση του οποίου:
    i. ελέγχει αν τα "data" του κόμβου όπου δείχνει ο $s2 ($18) είναι ή όχι μικρότερα από τον $s1 ($17),
    ii. αν είναι μικρότερα τα τυπώνει,
    iii. ελέγχει αν υπάρχει ή όχι επόμενος κόμβος στη λίστα,
    iv. αν δεν υπάρχει βγαίνει από το βρόγχο,
    v. αν υπάρχει, προχωρεί τον $s2 ($18) να δείξει σε αυτόν τον επόμενο κόμβο και επιστρέφει στην αρχή του βρόχου.
4. Μετά την έξοδο του βρόχου, επιστρέφει (πάντα) στην αρχή του δεύτερου μέρους του προγράμματος, γιά να ζητήσει μιά νέα τιμή και να ξανατυπώσει τα μικρότερα από αυτήν στοιχεία (εάν η τιμή δεν είναι αρνητική).

7.6: Χρήση υπορουτινών.

- Μετατρέψτε το βήμα 7.4(1) σε μια υπορουτίνα "read_int()" η οποία δεν παίρνει καμία παράμετρο εισόδου, διαβάζει έναν ακέραιο αριθμό, και επιστρέφει τον αριθμό που διάβασε.
- Μετατρέψτε το βήμα 7.4(3) σε μια υπορουτίνα "node_alloc()" η οποία δεν παίρνει καμία παράμετρο εισόδου, ζητάει από το σύστημα να δεσμεύσει ένα κόμβο για τη λίστα, και επιστρέφει τη διεύθυνση της μνήμης που έχει δεσμευθεί, ώστε το πρόγραμμα που καλεί τη node_alloc() να εισάγει τον κόμβο στη λίστα.
- Αλλάξτε το πρόγραμμα του 7.4 ώστε να χρησιμοποιεί τις ρουτίνες read_int() και node_alloc(). Στη χρήση καταχωρητών από τις διαδικασίες που γράφετε --εδώ από τις read_int() και node_alloc()-- όπως και από το πρόγραμμα σας που τις καλεί, πρέπει να τηρήσετε τις συμβάσεις χρήσης καταχωτητών (αν και, ειδικά οι read_int() και node_alloc() είναι αρκετά απλές και δεν είναι απαραίτητο να έχουν τοπικές μεταβλητές που να αποθηκεύονται στη στοίβα).
- Γράψτε μια υπορουτίνα print_node() που κάνει ό,τι περιγράφουν τα βήματα 7.5(3i) και 7.5(3ii). Η υπορουτίνα θα παίρνει ως εισόδους (α) τη διεύθυνση ενός κόμβου, και (β) τον ακέραιο προς τον οποίο συγκρίνουμε, και δεν θα επιστρέφει τίποτε.
- Γράψτε μια υπορουτίνα search_list() που υλοποιεί όλο το βήμα 7.5(3). Η ρουτίνα θα παίρνει ως πρώτη είσοδο τη διεύθυνση του πρώτου κόμβου τις λίστας (δείκτης σάρωσης) και σαν δεύτερη είσοδο την τιμή για την οποία πρέπει να ψάξει. Στη συνέχεια θα υλοποιεί τα βήματα (i,ii,iii,iv,v) καλώντας τη συνάρτηση print_node() για τα βήματα (i,ii). Όπως είπαμε, πρέπει να τηρείτε τις συμβάσεις χρήσης καταχωτητών· κάθε υπορουτίνα που χρειάζεται αποθήκευση κάποιου καταχωρητή ή/και της διεύθυνσης επιστροφής, θα πρέπει να έχει το δικό της stack frame.
- Αλλάξτε το πρόγραμμα σας του μέρους 7.5 ώστε να καλεί τις read_int() και search_list(), όπου η τελευταία θα καλεί την print_node(). Και πάλι πρέπει να χρησιμοποιήσετε τις συμβάσεις χρήσης των καταχωρητών μεταξύ των ρουτινών και του προγράμματος που τις καλεί. Επίσης η κάθε ρουτίνα που το χρειάζεται θα πρέπει να έχει το δικό της stack frame.
- Τελικά το συνολικό πρόγραμμα σας θα πρέπει να έχει μια main() που αποτελείται απο δύο μέρη. Το πρώτο μέρος της θα υλοποιεί το 7.4 με χρήση της node_alloc() και read_int(), ενώ το δεύτερο θα υλοποιεί το 7.5 με χρήση των read_int(), search_list(), και print_node(). Και πάλι, και η main() πρέπει να τηρεί τις συμβάσεις χρήσης καταχωρητών και του stack frame της (και να μπορεί και αυτή να επιστρέφει σε όποιον την κάλεσε (σε εκείνες τις "παράξενες" 9 εντολές του trap_handler.s που λέγαμε κάποτε...)).

Τρόπος Παράδοσης: Παραδώστε ηλεκτρονικά τον κώδικά σας, "**`ex07.s`**", κι ένα στιγμιότυπο της εκτέλεσής του στον SPIM, "**`ex07.jpg`**". Παραδώστε τα αυτά μέσω:
`turnin ex07@hy225 [directoryName]`
Θα εξεταστείτε ***και προφορικά*** για την Άσκηση 7, από βοηθό του μαθήματος, με διαδικασία γιά την οποία θα ενημερωθείτε μέσω ηλτά (email) στη λίστα του μαθήματος.
